
Lecture 12-2

Recursion

Binary Search: recursive implementation

```
/** Return the index of x in the sorted array a, or -1 if not found */
public static int find(int x, int[] a) {
    return find(x, a, 0, a.length - 1);
}

private static int find(int x, int[] a, int low, int high) {
    if (low > high)
        return -1;
    int med = (low + high) / 2;
    if (x == a[med])
        return med;
    else
        if (x < a[med])
            return find(x, low, med - 1);
        else
            return find(x, med + 1, high);
}
```

Integer.toString()

The task:

Convert integer values to strings.

For example, given the integer input 513,
output the string "513"

Recursive thinking:

- Base case: If $n < 10$ output " n "
- Reduction: an integer of the form $n = x\bar{y}yy \dots y$ (each x and y being a single digit) can be reduced using $x = n/10$ and $\bar{y}yy \dots y = n \% 10$
- Assembly: the concatenation operator $+$ can be used to combine partial results into the final return value.

`Integer.toString()`: recursive implementation

The task:

Convert integer values to strings.

For example, given the integer input 513,
output the string "513"

```
public class PrintIntegerDemo {  
  
    public static void main(String[] args) {  
        System.out.print(toString(513));  
    }  
  
    static String toString (int n) {  
        if (n < 10)  
            return "" + n;  
        else  
            return toString(n / 10) + n % 10;  
    }  
}
```

`toString(5137) = toString(513) + '7'`

'513' |
 |
 |

`toString(51) + '3'`

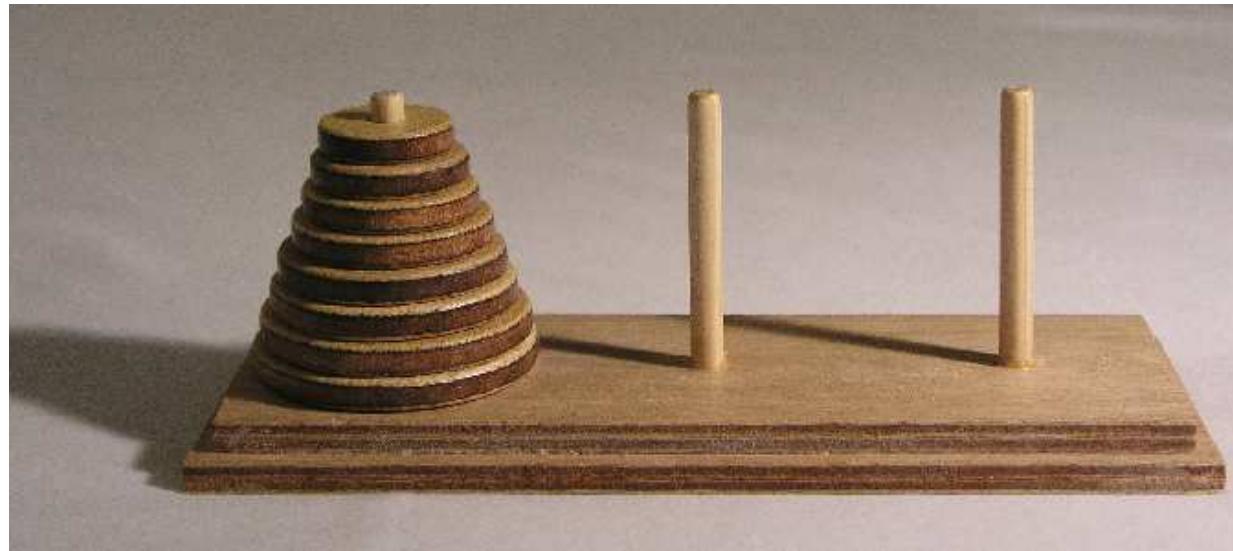
'51' |
 |
 |

`toString(5) + '1'`

'5' |
 |
 |

Base case: $5 < 10$

Tower of Hanoi



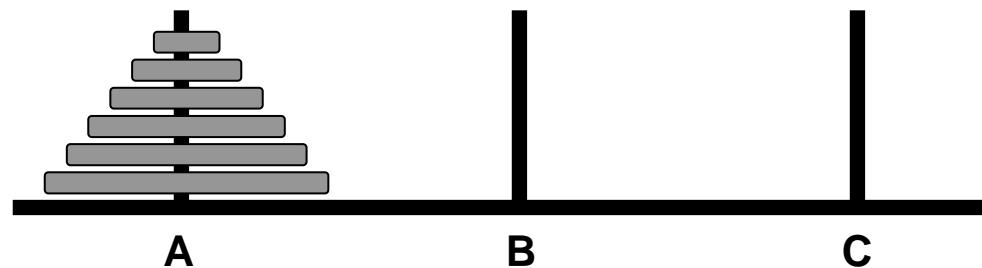
The rules of the game:

Move all the disks to another spindle, so that:

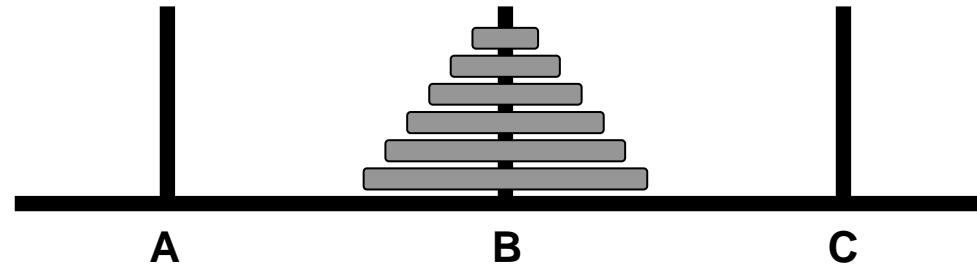
- (1) only one disk moves at a time
- (2) The smaller disks are always at the top

Tower of Hanoi

Start:



Finish:



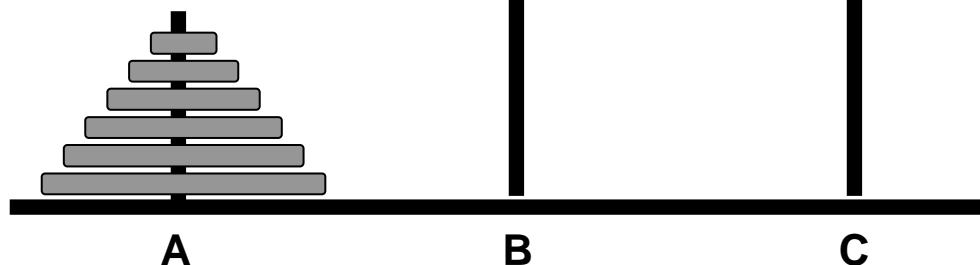
The rules of the game:

Move all the disks to another spindle, so that:

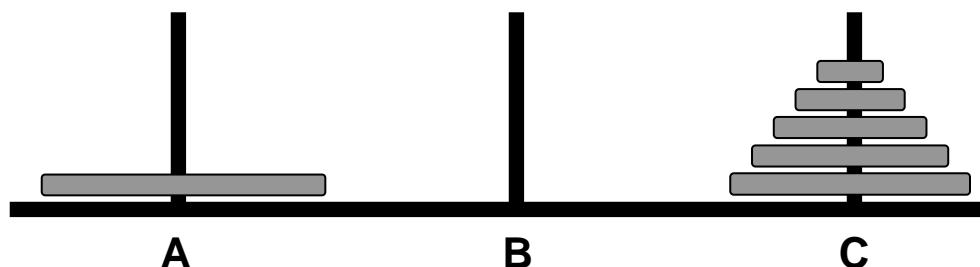
- (1) only one disk moves at a time
- (2) The smaller disks are always at the top

The recursive strategy

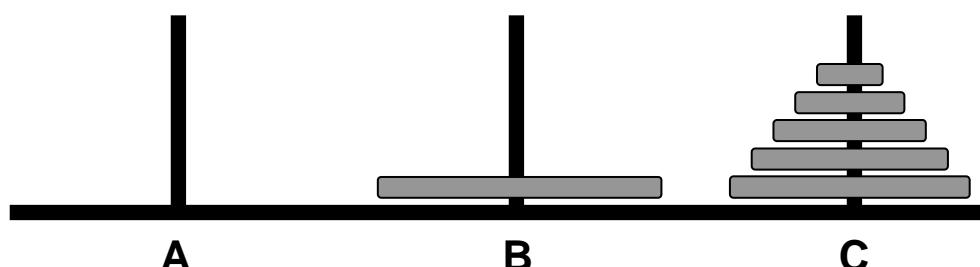
Start:



Move
the top $n-1$
disks from
A to C

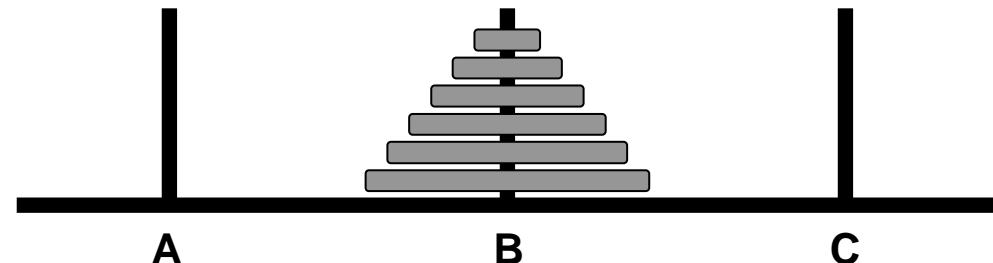


Move
the remaining
disk from
A to B



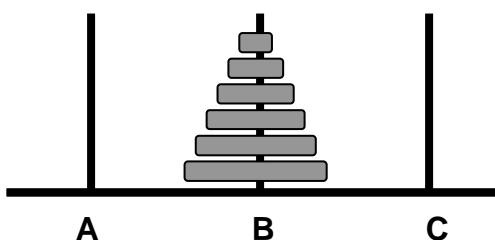
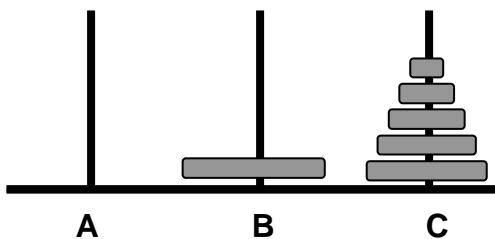
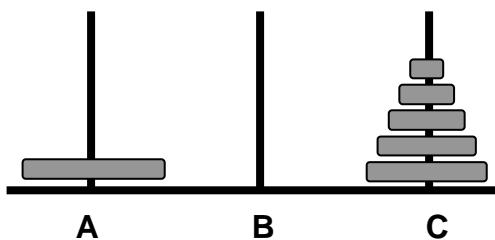
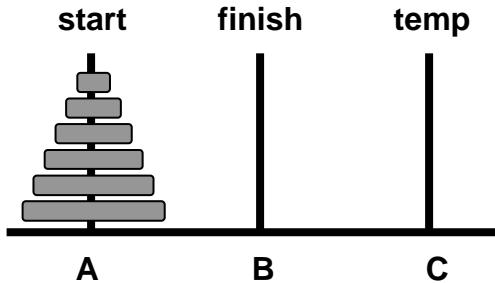
Move
all the disks
from
C to B

Finish:



Done!

Tower of Hanoi implementation



```
if n = 1 : move this single disk from start to finish  
if n > 1 :  
    1. Move the top n-1 disks from start to temp, using finish  
    2. Move the bottom disk from start to finish  
    3. Move the top n-1 disks from temp to finish, using start
```

```
public class TowerOfHanoi {  
    public static void main(String[] args) {  
        moveTower(3, "A", "B", "C");  
    }  
  
    // Moves a tower of disks from A to B using C  
    static void moveTower(int n, String start,  
                          String finish, String temp) {  
        if (n==1)  
            System.out.println(start + "->" + finish);  
        else {  
            moveTower(n-1, start, temp, finish);  
            System.out.println(start + "->" + finish);  
            moveTower(n-1, temp, finish , start);  
        }  
    }  
}
```

A terminal window titled 'C:\WINDOWS\system32' shows the command 'D:\demo>java TowerOfHanoi' followed by the sequence of moves: A->B, A->C, B->C, A->B, C->A, C->B, and A->B.

Simulation

```
moveTower(3, "A", "B", "C")
```

Goal: move a tower of size 3 from A to B using C:

Goal: move a tower of size 2 from A to C using B:

Goal: move a tower of size 1 from A to B using C: ←

This

mov

Sys

mov

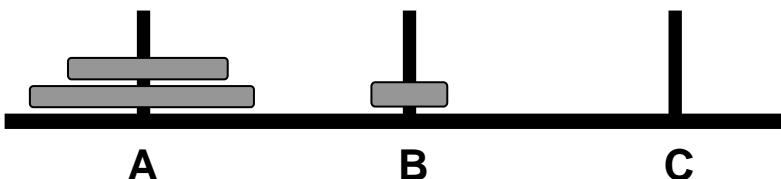
Sys

mov

n	start	finish	temp
1	"A"	"B"	"C"

Things to do:

```
moveTower(n-1, start, temp, finish);  
System.out.println(start + "-" + finish);  
moveTower(n-1, temp, finish , start);
```



Simulation

```
moveTower(3, "A", "B", "C")
```

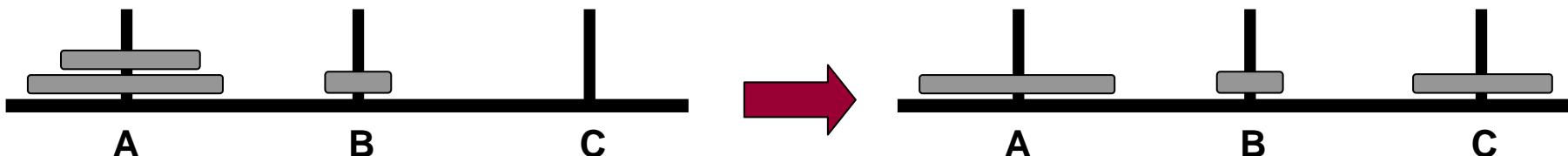
Goal: move a tower of size 3 from A to B using C:

Goal: move a tower of size 2 from A to C using B:

n	2	start	"A"	finish	"C"	temp	"B"
---	---	-------	-----	--------	-----	------	-----

This
move
System
move

Things to do:
~~moveTower(n-1, start, temp, finish);~~
System.out.println(start + "->" + finish); ←
moveTower(n-1, temp, finish, start);



Simulation

```
moveTower(3, "A", "B", "C")
```

Goal: move a tower of size 3 from A to B using C:

Goal: move a tower of size 2 from A to C using B:

n	2	start	"A"	finish	"C"	temp	"B"
---	---	-------	-----	--------	-----	------	-----

This

mov

Sys

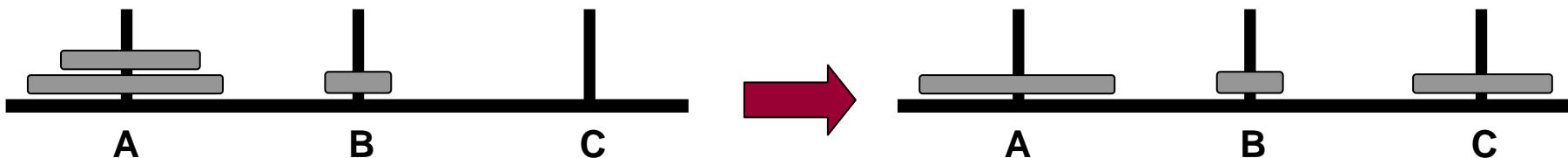
mov

Things to do:

~~moveTower(n-1, start, temp, finish);~~

~~System.out.println(start + " > " + finish);~~

moveTower(n-1, temp, finish, start);



Simulation

```
moveTower(3, "A", "B", "C")
```

Goal: move a tower of size 3 from A to B using C:

Goal: move a tower of size 2 from A to C using B:

n	2	start	"A"	finish	"C"	temp	"B"
---	---	-------	-----	--------	-----	------	-----

This

mov

Sys

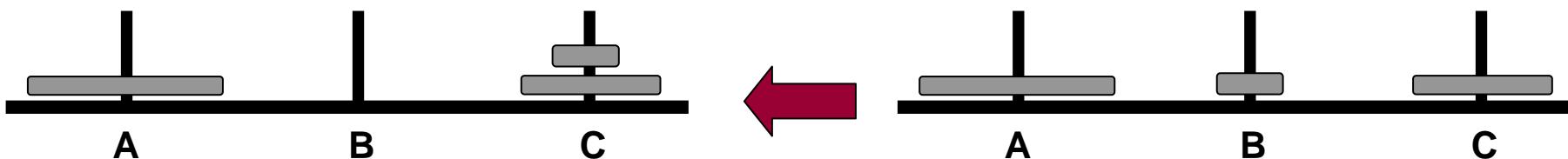
mov

Things to do:

~~moveTower(n-1, start, temp, finish);~~

~~System.out.println(start + " > " + finish);~~

moveTower(n-1, temp, finish, start);



Permutations

The task:

Write a method `listPermutations(String s)` that generates a complete set of all the permutations of the characters in `s`.

For example, `listPermutations("abcd")` will generate:

abcd	bacd	cabd	dabc
abdc	badc	cadb	dacb
acbd	bcad	cbad	dbac
acdb	bcda	cbda	dbca
adbc	bdac	cdab	dcab
adcb	bdca	cdba	dcba

(one after the other)

Observations:

- If `length(s) = n`, there are $n * (n-1) * (n-2) * \dots * 1 = n!$ permutations
- the list of permutations starting with "a" contains the prefix "a" followed by all the permutations of "bcd"

Strategy

Given "abcd" we have to generate:

abcd	bacd	cabd	dabc
abdc	badc	cadb	dacb
acbd	bcad	cbad	dbac
acdb	bcda	cbda	dbca
adbc	bdac	cdab	dcab
adcb	bdca	cdba	dcba

Print 'a' + all permutations of "abcd" without charAt 0

Print 'b' + all permutations of "abcd" without charAt 1

Print 'c' + all permutations of "abcd" without charAt 2

Print 'd' + all permutations of "abcd" without charAt 3

```
C:\WINDOWS\system32\cmd.exe
D:\demo>java ListPerm
abcd
abdc
acbd
acdb
adbc
adcb
bacd
badc
bcad
bcda
bdac
bdca
cabd
cadb
cbad
cbda
cdab
cdba
dabc
dacb
dbac
dbc
dcab
dcba
```

```
// String s without charAt(i):
rest = s.substring(0,i) + s.substring(i + 1 , s.length());
```

Implementation

```
public class ListPermutationsDemo {  
  
    public static void main(String[] args) {  
        listPermutations("abcd");  
    }  
  
    public static void listPermutations(String s) {  
        listPermutations("", s);  
    }  
  
    private static void listPermutations (String prefix, String s) {  
        if (s.length() == 0)  
            System.out.println(prefix);  
        else  
            for (int i = 0; i < s.length(); i++) {  
                char ch = s.charAt(i);  
                String rest = s.substring(0,i) + s.substring(i+1);  
                listPermutations(prefix + ch, rest);  
            }  
    }  
}
```

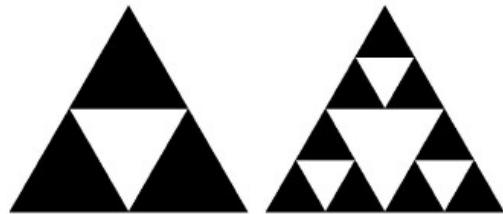


```
C:\WINDOWS\system32\cmd  
D:\demo>java ListPermutations  
abcd  
abdc  
acbd  
acdb  
adbc  
adcb  
adch  
bacd  
badc  
bcad  
bcda  
bdac  
bdca  
cabd  
cadb  
cbad  
cbda  
cdab  
cdba  
dabc  
dacb  
dbac  
dbc a  
dcab  
dcba
```

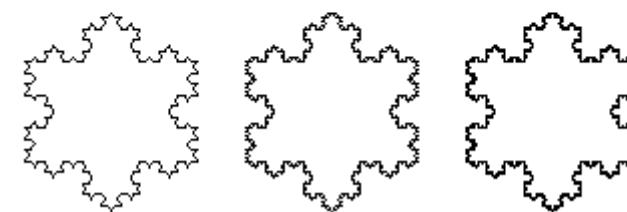
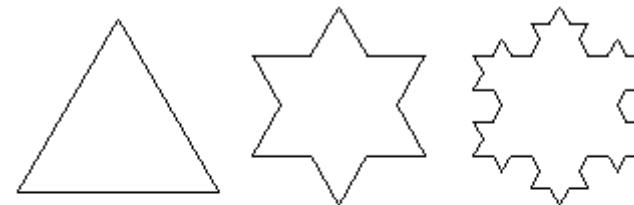
Fractals

Fractal: A geometric shape that can be split into parts, each of which is (either exactly or approximately) a reduced-size copy of the whole

Some well-known fractal shapes:



Sierpinski triangles

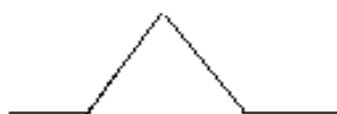


Koch snowflake

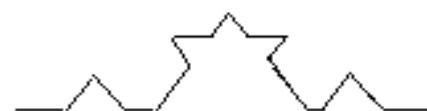
Typical generative strategy:



Level 0

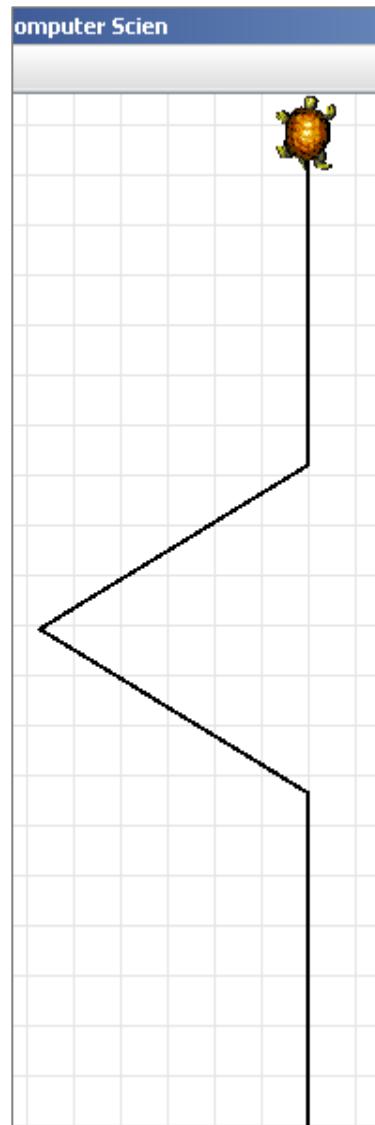


Level 1

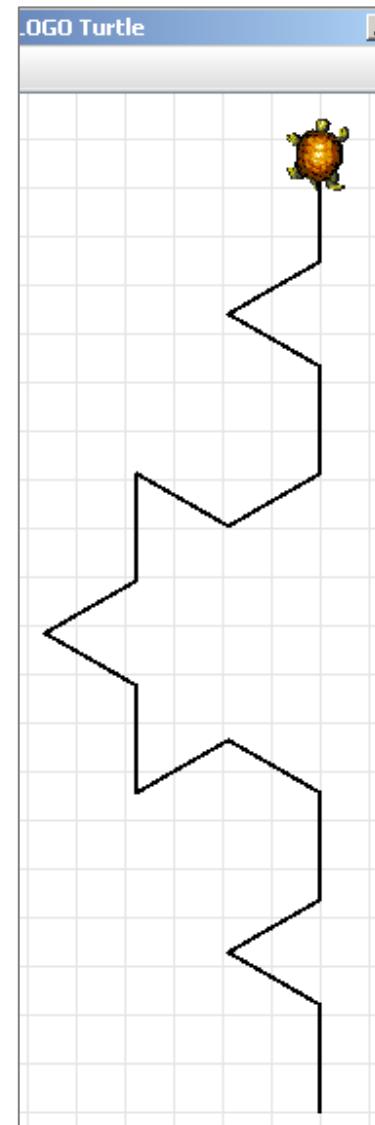


Level 2

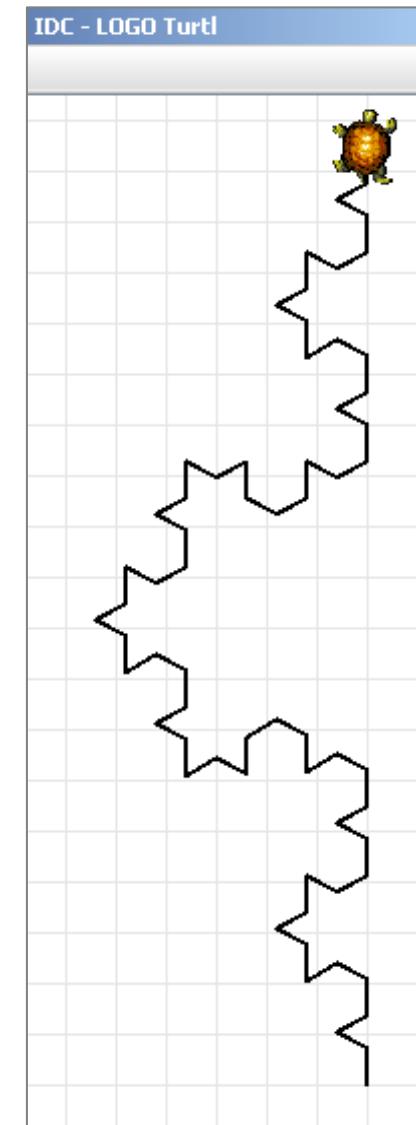
TurtleFractal



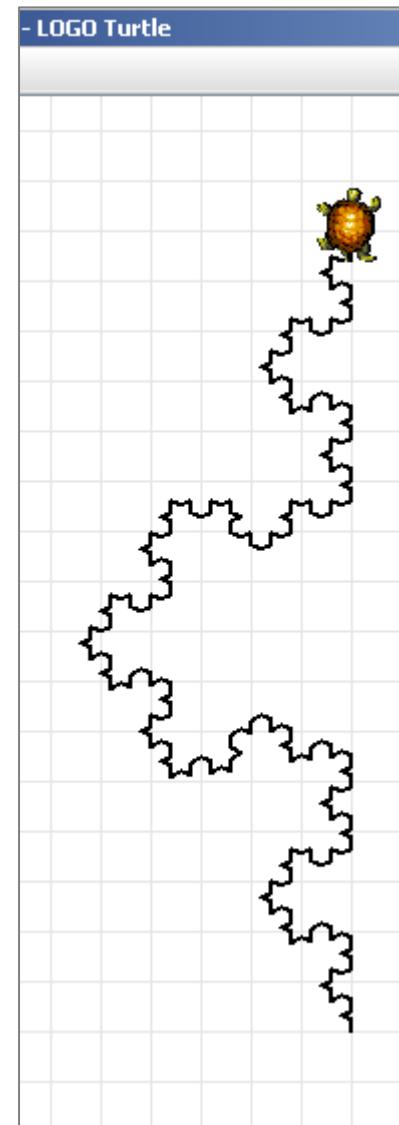
DrawFractal(500 , 1)



DrawFractal(500 , 2)



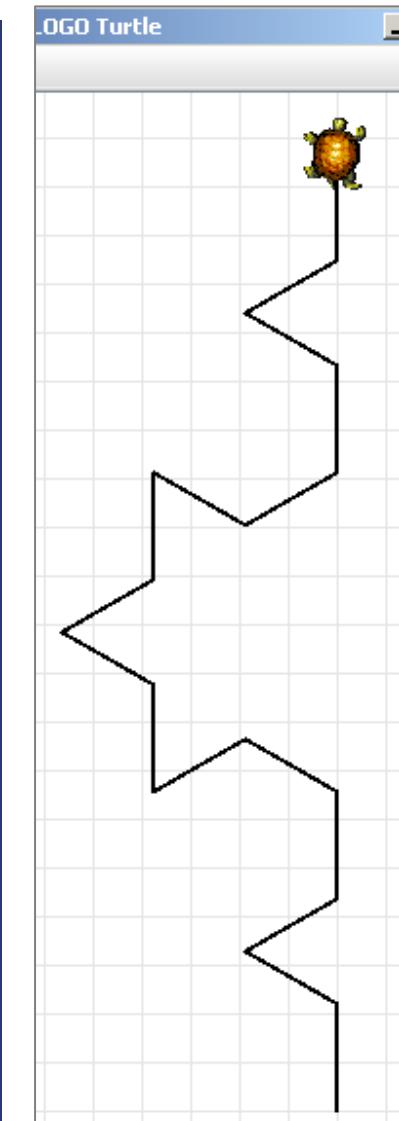
DrawFractal(500 , 3)



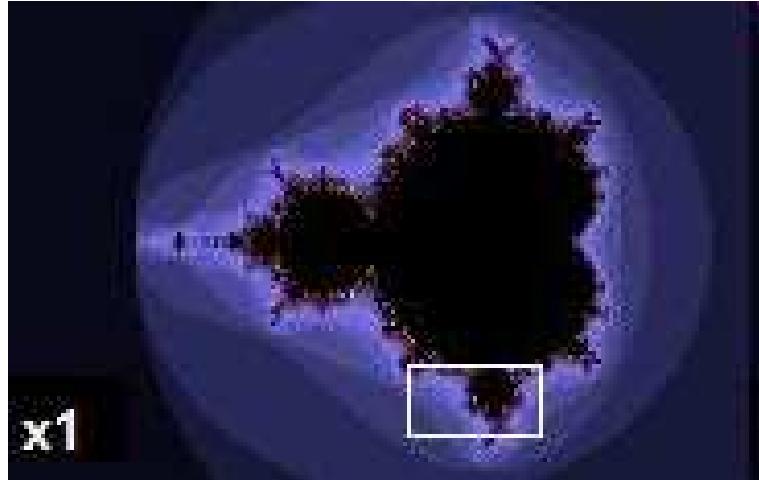
DrawFractal(500 , 4)

Implementation

```
public class TurtleFractal {  
    static Turtle turtle = new Turtle();  
    public static void main(String[] args) {  
        turtle.tailDown();  
        drawFractal(500,2);  
    }  
  
    public static void drawFractal(int length, int level) {  
        if (level==0)  
            turtle.moveForward(length);  
        else {  
            drawFractal(length/3, level-1);  
            turtle.turnLeft(60);  
            drawFractal(length/3, level-1);  
            turtle.turnRight(120);  
            drawFractal(length/3, level-1);  
            turtle.turnLeft(60);  
            drawFractal(length/3, level-1);  
        }  
    }  
}
```



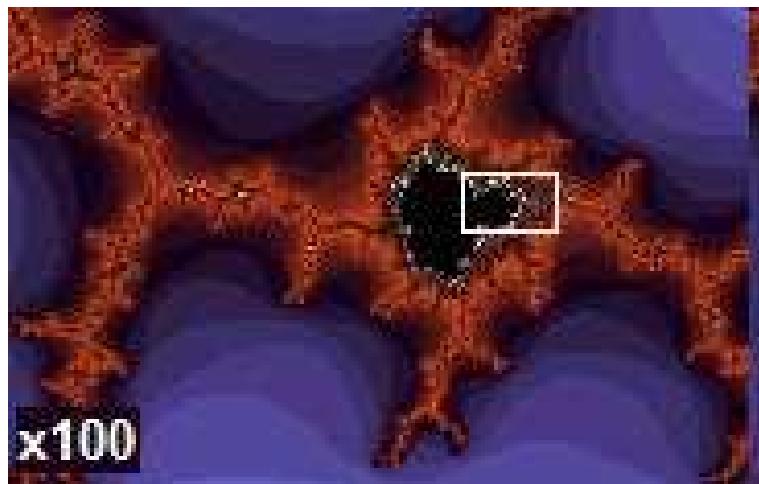
Fractals



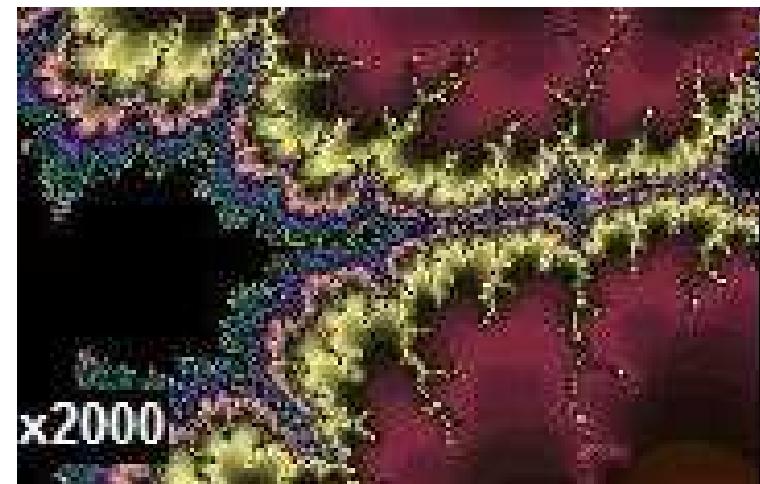
x1



x6



x100



x2000